ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2006
Seria: AUTOMATYKA z. 144 Nr kol. 1727

Józef GRABOWSKI, Jarosław PEMPERA
Politechnika Wrocławska

MINIMALIZACJA SUMY CZASÓW ZAKOŃCZENIA REALIZACJI ZADAŃ W SYSTEMIE PRZEPŁYWOWYM Z OGRANICZENIEM „BEZ CZEKANIA"

Streszczenie. W pracy przedstawiono algorytmy oparte na metodzie przeszukiwania zstępującego wykorzystujące mechanizm jednoczesnego wykonywania wielu ruchów elementarnych. Z rezultatów testów obliczeniowych przeprowadzonych na instancjach Tailarda wynika, że zaproponowany mechanizm pozwala na generowanie lepszych rozwiązań przy wykonywaniu znacznie mniejszej liczby iteracji przez algorytm zstępujący.

MINIMIZING TOTAL COMPLETION TIME IN NO-WAIT FLOW-SHOP SCHEDULING PROBLEM

Summary. This paper deals with a descending search algorithms for the no-wait flow-shop problem. In the algorithms the multimoves are used that consist in performing several moves simultaneously in a single iteration of algorithm. The proposed algorithms is empirically evaluated on the Tailard's benchmarks.

1. Wstęp

W ostatnich latach obserwuje się duże zainteresowanie problemami szeregowania zadań z trudnymi kryteriami optymalizacyjnymi, jakimi są kryteria „sumacyjne". Już najprostsze, dwumaszynowe problemy z kryterium minimalizacji sumy czasów zakończenia realizacji zadań są problemami NP-trudnymi [5]. Jednakże duża ilość metod konstruowania algorytmów przybliżonych opartych na metodach przeszukiwań lokalnych pozwala na konstruowanie algorytmów heurystycznych, które generują dobrej jakości rozwiązania heurystyczne w czasie akceptowalnym przez planistów.

Spośród wielu algorytmów opracowanych dla problemu przepływowego z ograniczeniem „bez czekania" oraz omawianym kryterium należy wymienić algorytmy konstrukcyjne [6], algorytmy iteracyjne oparte na przeszukiwaniu genetycznym [2], mrówkowym [7] oraz metodzie wstawień [1]. Proces obliczeniowy większości tych algorytmów wspomagany jest działaniem algorytmu zstępującego. Jego efektywność ma duże znaczenie przede wszystkim dla szybkości znajdowania dobrej jakości rozwiązań końcowych. Własności problemu pozwalające na wyliczenie wartości funkcji celu w czasie O(1) sprawiają, że te algorytmy są wyjątkowo szybkie.

2. Sformułowanie problemu

W problemach przepływowych należy wykonać zbiór składający się z n zadań na m maszynach. Każde zadanie jest wykonywane kolejno na maszynach $1,2,\ldots,m$. Każda maszyna może wykonywać tylko jedno zadanie w danej chwili, a wykonywanie zadań nie może być przerywane. Zatem każde zadanie j składa się z ciągu m operacji; k–ta operacja odpowiada wykonywaniu zadania na maszynie k w czasie $p_{jk}>0$.

W problemach z ograniczeniem „bez czekania" wykonywanie kolejno następujących po sobie operacji należących do tego samego zadania musi następować bez zwłoki. Ograniczenie to narzuca jednakową kolejność wykonywania zadań na wszystkich maszynach [4]. Zatem kolejność wykonywania zadań możemy opisać za pomocą permutacji elementów zbioru $\{1,\ldots,n\}$.

Niech Π oznacza zbiór wszystkich permutacji określonych na zbiorze $\{1,\ldots,n\}$. W rozważanym problemie należy znaleźć permutację $\pi^* \in \Pi$ taką, że

$$C_{sum}(\pi^*) = \min_{\pi\in\Pi} C_{sum}(\pi^*),$$

gdzie $C_{sum}(\pi) = \sum_{j=1}^{n} C_j$, natomiast C_j oznacza czas zakończenia realizacji ostatniej operacji zadnia j.

Niech zadania i oraz j będą bezpośrednio po sobie następującymi zadaniami („i przed j") w pewnej kolejności wykonywania zadań. Wówczas przez $d_{i,j}^k = c_{i,j} - a_i^k - b_i^k$, gdzie $c_{i,j} = \max_{1\le s\le m}(a_i^s - b_i^s)$, $a_j^k = \sum_{s=1}^{k} p_{j,s}$, $b_j^k = \sum_{s=k}^{m} p_{j,s}$, definiujemy minimalne opóźnienie czasu rozpoczęcia wykonywania k-tej operacji zadania j względem czasu zakończenia realizacji k-tej operacji zadania i wynikające z ograniczenia „bez czekania". Opóźnienie to nie zależy od kolejności wykonywania zadań poprzedzających parę zadań (i, j) oraz zadań po niej następujących.

Dla ustalonej kolejności wykonywania zadań π czasy zakończenia i / lub rozpoczęcia wykonywania zadań możemy wyznaczyć z następującego równania rekurencyjnego:

$$C_{\pi(j)} = C_{\pi(j-1)} + d_{\pi(j-1),\pi(j)}^m + p_{\pi(j),m},$$

gdzie $d_{\pi(0),\pi(j)}^m = \sum_{s=1}^{m-1} p_{j,s}$, które następnie można rozwinąć do postaci:

$$C_{\pi(j)} = \sum_{s=1}^{j} (d_{\pi(s-1),\pi(s)}^m + p_{\pi(s),m}).$$

Dalej, przyjmując $D_{i,j} = d_{i,j}^m + p_{j,m}$, otrzymujemy $C_{\pi(j)} = \sum_{s=1}^{j} D_{\pi(s-1),\pi(s)}$ oraz:

$$C_{sum}(\pi) = \sum_{j=1}^{n} C_j = \sum_{j=1}^{n}\sum_{s=1}^{j} D_{\pi(s-1),\pi(s)} = \sum_{j=1}^{n}(n+1-j)D_{\pi(j-1),\pi(j)}.$$

3. Algorytmy zstępujące oparte na ruchach złożonych

Algorytmy zstępujące należą do najprostszych algorytmów należących do klasy algorytmów lokalnej optymalizacji. W każdej iteracji tego typu algorytmu dla pewnego rozwiązania x, zwanego rozwiązaniem bazowym, generowany jest w sposób uporządkowany zbiór rozwiązań sąsiednich zwanych sąsiedztwem $N(x)$. Rozwiązania

sąsiednie powstają z rozwiązania bazowego przez niewielką jego modyfikację, zwaną ruchem. Z sąsiedztwa wybiera się rozwiązanie najlepsze (reguła „największej poprawy") lub pierwsze rozwiązanie, które jest lepsze od rozwiązania bazowego (reguła „pierwszej poprawy"). Wybrane rozwiązanie staje się rozwiązaniem bazowym w następnej iteracji algorytmu. Algorytm kończy działanie, gdy w otoczeniu rozwiązania bazowego nie znajdują się rozwiązania lepsze od niego.

W problemach, w których rozwiązaniem jest permutacja, najczęściej stosowane ze względu na dużą skuteczność, są ruchy typu *wstaw* oraz ruchy typu *zamień*. W obu przypadkach opisywane są one za pomocą pary (x,y) pozycji w permutacji. Para $v=(x,y)$ definiuje ruch typu *wstaw* (Ins) w permutacji π, który usuwa zadanie $\pi(x)$ z pozycji x w permutacji π, a następnie wstawia je na pozycję y w tej permutacji. Sąsiedztwo generowane przez ten typ ruchów składa się z $(n\text{-}1)^2$ rozwiązań. Para $v=(x,y)$ definiuje ruch typu *zamień* (Ich) w permutacji π, który usuwa zadanie $\pi(x)$ z pozycji x oraz zadanie $\pi(y)$ z pozycji y, a następnie wstawia zadanie $\pi(y)$ na pozycję x, natomiast zadanie $\pi(x)$ na pozycję y. Sąsiedztwo generowane przez ten typ ruchów składa się z $n(n\text{-}1)/2$ różnych rozwiązań. Podzbiorem ruchów typu zamień są ruchy typu *zamień* sąsiednie, w których zamianie podlegają zadania stojące na dwóch kolejnych pozycjach w permutacji π.

Dla problemu przepływowego z ograniczeniem „bez czekania" oraz kryterium minimalizacji czasu zakończenia realizacji wszystkich zadań można ze zbioru wszystkich ruchów poprawiających wybrać podzbiór ruchów, których jednoczesne wykonanie spowoduje wygenerowanie rozwiązania, dla którego poprawa wartości funkcji celu względem rozwiązania bazowego równa będzie sumie popraw uzyskiwanych przez każdy z ruchów składowych [3]. Sposób konstruowania takiego zbioru ruchów (multiruchu) opiera się na zbiorze ruchów poprawiających $PZ=\{v\in N(\pi)\,|\,C_{\text{sum}}(\pi_v)<C_{\text{sum}}(\pi)\}$, z którego wybiera się podzbiór ruchów niezależnych, w którym dla każdej pary ruchów spełniony jest co najmniej jeden z warunków:

$$(1)\begin{cases}\max(x_1,y_1)+1<\max(x_2,y_2)\\ \text{lub}\\ \max(x_2,y_2)+1<\max(x_1,y_1).\end{cases}$$

$$(2)\begin{cases}\min(x_1,y_1)+1<\min(x_2,y_2)\ \textit{oraz}\ \max(x_2,y_2)+1>\max(x_1,y_1)\\ \text{lub}\\ \min(x_2,y_2)+1<\min(x_1,y_1)\ \textit{oraz}\ \max(x_1,y_1)+1>\max(x_2,y_2).\end{cases}$$

$$(3)\begin{cases}\min(x_1,y_1)+1<\min(x_2,y_2)\ \textit{oraz}\ \min(x_2,y_2)+1<\max(x_1,y_1)\ \textit{oraz}\ \max(x_2,y_2)+1<\max(x_1,y_1)\\ \text{lub}\\ \min(x_2,y_2)+1<\min(x_1,y_1)\ \textit{oraz}\ \min(x_1,y_1)+1<\max(x_2,y_2)\ \textit{oraz}\ \max(x_1,y_1)+1<\max(x_2,y_2).\end{cases}$$

Można pokazać, że dla rozważanego w pracy kryterium optymalizacyjnego C_{sum} zbiór ruchów poprawiających, w którym dla każdej pary ruchów spełniony jest warunek (1), ma taką samą cechę dotyczącą sumowania się popraw generowanych przez ruchy składowe.

4. Badania testowe

W celu zbadania efektywności ruchów złożonych oprogramowano w języku C++ kilka wariantów algorytmu zstępującego opartego na regule największej poprawy. Poszczególne wersje algorytmów różniły się rodzajem otoczenia, Ins – wstaw, Ich – zamień, Ins&Ich – otoczenie będące sumą otoczeń wstaw i zamień oraz zastosowanym sposobem konstruowania multiruchu, DS(T1) – spełniający tylko warunek (1), DS(T1-3) – spełniający jeden z warunków (1)-(3), DS – bez multiruchu.

Test został przeprowadzony na dziewięciu pierwszych grupach przykładów Tailarda [8]. Każda z tych grup składa się z 10 instancji problemu przepływowego. Rozwiązania generowane przez algorytmy porównano z rozwiązaniem generowanym przez algorytm AA, zaczerpnięty z pracy [1], który został uruchomiony na 10 iteracji.

Testowanie algorytmów przebiegało w dwóch etapach. Pierwszy polegał na porównaniu efektywności różnych wariantów algorytmu zstępującego. Każdy z wariantów algorytmu został uruchomiony jeden raz z rozwiązania generowanego, podobnie jak w pracy [1], prostym algorytmem konstrukcyjnym, którego opis znajduje się również w tej pracy. Drugi etap miał na celu porównanie najlepszego z algorytmów DS z algorytmem AA. W tym celu każdy z testowanych algorytmów dla każdej instancji uruchomiono 10 razy z rozwiązania wygenerowanego w sposób losowy. Dodatkowo dla każdej instancji wygenerowano 1000 rozwiązań losowych – algorytm RAND.

Dla każdego rozwiązania π wygenerowanego przez testowane algorytmy wyznaczono błąd względny w następujący sposób: $B(\pi)=100\%(C_{\mathrm{sum}}(\pi)-C^{Ref})/C^{Ref}$, gdzie C^{Ref} jest wartością funkcji C_{sum} dla rozwiązania wygenerowanego algorytmem AA. Oczywiście, wartość dodatnia $B(\pi)$ oznacza, że rozwiązanie π jest gorsze od rozwiązania wygenerowanego algorytmem AA, wartość ujemna oznacza, że lepsze.

W tabelach 1, 2 i 3 przedstawiono uśrednione wyniki badań eksperymentalnych. W przypadku tabeli 1 i 2 uśrednienie dotyczyło wyników w obrębach grup przykładów, natomiast w przypadku tabeli 3 w obrębie zbioru rozwiązań wygenerowanych dla tej samej instancji (algorytmy uruchamiane były wielokrotnie z różnych rozwiązań początkowych) oraz tak samo jak w przypadku tabel 1 i 2.

Z prezentowanych w tabelach 1 i 2 rezultatów testu wynika, że najlepsze rozwiązania generują algorytmy wykorzystujące otoczenie mieszane Ins&Ich, w dalszej kolejności algorytmy oparte na otoczeniu Ins oraz zdecydowanie najgorsze algorytmy oparte na otoczeniu Ich. Porównując algorytmy wykorzystujące różne sposoby generowania multiruchu możemy zauważyć, że w obrębie tego samego otoczenia wartości funkcji celu są porównywalne, jednakże biorąc pod uwagę liczbę iteracji wykonywanych przez algorytm (czas działania) zdecydowanie najlepszym algorytmem jest algorytm oparty na multiruchach T1-3. Wykonuje on nawet prawie 5 razy mniej iteracji niż klasyczny algorytm DS.

Tabela 1

Uśredniony błąd względny algorytmów

Grupa	DS.			DS(T1)			DS(T1-3)		
	Ins&Ich	Ins	Ich	Ins&Ich	Ins	Ich	Ins&Ich	Ins	Ich
20×5	-0,63	1,29	2,15	-0,51	1,26	1,97	-0,85	1,07	1,94
20×10	2,27	2,61	4,31	2,21	2,71	4,40	2,43	3,46	4,35
20×20	2,22	4,80	4,48	2,10	4,76	4,51	2,13	4,22	4,24
50×5	-0,45	1,22	1,57	-0,60	1,10	1,33	-0,56	0,94	1,18
50×10	0,72	2,49	4,31	1,10	2,93	4,19	1,07	2,48	3,59
50×20	2,09	4,01	5,63	2,36	4,09	4,68	2,04	3,72	5,04
100×5	-0,74	1,08	1,60	-0,23	0,87	1,84	-0,75	1,04	1,44
100×10	0,55	2,47	3,94	0,65	2,34	3,99	0,95	2,51	4,38
100×20	2,17	3,13	5,15	1,93	2,89	5,72	1,47	3,10	4,91
Średnio	0,91	2,57	3,68	1,00	2,55	3,63	0,88	2,51	3,45

Tabela 2

Uśredniona liczba iteracji algorytmów

Grupa	DS.			DS(T1)			DS(T1-3)		
	Ins&Ich	Ins	Ich	Ins&Ich	Ins	Ich	Ins&Ich	Ins	Ich
20×5	7	6	6	4	4	4	3	3	3
20×10	6	6	6	5	4	4	3	4	3
20×20	8	6	8	5	4	4	4	3	4
50×5	20	16	17	9	6	8	5	5	7
50×10	21	20	15	11	11	8	6	6	6
50×20	23	22	18	11	11	9	6	7	8
100×5	47	35	42	15	11	12	9	7	9
100×10	47	42	34	18	17	15	8	7	8
100×20	48	46	41	20	18	13	10	8	10

Tabela 3

Wyniki porównania różnych algorytmów

Grupa	RAND		AA		DS(T1-3)			AA+DS(T1-3)		
	AVE	AMIN	AVE	AMIN	AVE	AMIN	ITER	AVE	AMIN	ITER
20×5	41,54	18,53	-0,34	-1,29	-0,21	-1,93	6	-1,25	-1,84	2
20×10	39,36	17,07	0,52	-0,72	1,18	-0,94	5	-0,20	-1,01	2
20×20	33,05	15,19	0,54	-0,51	1,68	-0,44	5	0,53	-0,60	2
50×5	54,14	37,39	0,12	-1,27	0,49	-1,28	5	-1,40	-2,31	1
50×10	53,87	37,95	0,08	-1,06	1,11	-0,98	6	-0,73	-1,82	1
50×20	52,33	35,66	0,12	-0,93	1,38	-0,65	6	-0,11	-1,60	0
100×5	61,26	48,85	0,45	-0,48	0,36	-0,97	6	-1,69	-2,50	1
100×10	62,52	49,02	0,01	-0,82	1,36	-0,34	5	-1,11	-2,16	1
100×20	62,28	49,17	0,52	-0,45	1,95	0,31	7	-0,37	-1,27	1
Średnio	51,15	34,31	0,22	-0,84	1,03	-0,80		-0,70	-1,68	

Analizując wyniki prezentowane w tabeli 3 możemy zauważyć, że znalezienie dobrych rozwiązań w sposób przypadkowy jest bardzo trudne, nawet dla instancji o małej liczbie zadań. Algorytm DS(T1-3) startując z rozwiązań losowych generuje rozwiązania gorsze w sensie średnim niż algorytm AA, lecz biorąc pod uwagę wartość najmniejszą z kilku uruchomień wyniki dla obu algorytmów są porównywalne.

Zdecydowanie najlepsze rezultaty osiągnięto poprawiając rozwiązanie wygenerowane przez algorytm AA algorytmem zstępującym DS(T1-3).

LITERATURA

1. Aldowaisan T., Allahverdi A.: New heuristics for m-machine no-wait flowshop to minimize total completion time. Omega 32, 2004, p. 345-352.
2. Chen C.L., Neppali R.V., Aljaber N.: Genetic algorithms applied to the continuous flow shop problem. Computers and Industrial Engineering 30, 1996, p. 757-765.
3. Grabowski J., Pempera J.: Some local search algorithms for no-wait flow-shop problem with makespan criterion. Computers and Operations Research 92, 2005, p. 2197-2212.
4. Grabowski J., Pempera J.: Sequencing of jobs in some production system. European Journal of Operational Research 125, 2000, p. 535-550.
5. Hall N.G., Sriskandarajah C.: A survey of machine scheduling problems with blocking and no-wait in process. Operations research 44, 1996, p. 510-525.
6. Rajedran C., Chaudhuri D.: Heuristic algorithms for continuous flow–shop problem. Naval Research Logistics 37, 1996, p. 695-705.
7. Shyu S.J., Lin B.M.T., Yin P.Y.: Application of ant colony optimization for no-wait flowshop scheduling problem to minimize total completion time. Computers and industrial engineering 47, 2004, p. 181-193.
8. Taillard E.: Benchmarks for basic scheduling instances. European Journal of Operational Research, 64, 1993, p. 278-285.

Recenzent: Prof. dr hab. inż. Konrad Wala

Abstract

This paper deals with a descending search algorithms for the no-wait flow-shop problem. In the algorithms the multimoves are used that consist in performing several moves simultaneously in a single iteration of algorithm The proposed algorithm is empirically evaluated on the Tailard's benchmarks and found to be relatively more effective in finding better solutions in a much shorter time than classical descending search algorithms. The proposed algorithms can be used for the construction advanced metaheuristics.